Framework orientado a la generación de aplicaciones gráficas integrando Realidad Aumentada y gráficos de alta complejidad

José Ignacio Schneider[1,2], Sergio Martig[1], Silvia Castro[1]

[1] Laboratorio de Investigación y Desarrollo en Visualización y Computación Gráfica
Departamento de Ciencias e Ingeniería de la Computación - Universidad Nacional del Sur
Av. Alem 1253 – 8000 – Bahía Blanca – Buenos Aires – Argentina
[2] Comisión de Investigaciones Científicas de la Pcia. de Bs Aires
{jis, srm, smc}@cs.uns.edu.ar

La Realidad Aumentada (RA) tiene un gran auge en el mundo gráfico actual. Sin embargo, es un área de la Computación Gráfica que aún no ha madurado en muchos de sus aspectos. Hoy en día, hay aún pocos *frameworks* que permiten crear aplicaciones de RA de manera sencilla incorporando las últimas tecnologías gráficas. Esto motivó la creación de un *framework* de desarrollo simple e intuitivo para la realización de aplicaciones de RA, en el que se integran distintas técnicas de RA sobre un motor gráfico extensible que provee tecnologías gráficas avanzadas.

Palabras clave: Realidad Aumentada, Computación Gráfica, Dispositivos No Convencionales, Visualización.

1 Introducción

La RA es un área de la computación gráfica que trabaja combinando o embebiendo elementos sintéticos o datos generados por computadora en el mundo real a través de algún sistema de display y de dispositivos adecuados de interacción.

La RA está emergiendo como una tecnología con grandes posibilidades de aplicación en diferentes ámbitos. Nuevas alternativas plantean conducir hacia aplicaciones de RA novedosas, flexibles y de gran eficiencia en muchos dominios de aplicación tanto móviles como no, habilitando nuevas áreas de aplicación para museos, educación, entretenimiento, investigación, industria, arte, etc.

En los últimos años, un número creciente de empresas han desarrollado aplicaciones de RA con propósitos comerciales reales. Paulatinamente se está accediendo a esta tecnología y a aplicaciones que la explotan. Ejemplos de esta evolución comercial son los trabajos realizados por empresas como SONY, Microsoft y Nintendo, que han desarrollado hardware y software con el propósito de crear aplicaciones de RA innovadoras. Como ejemplos concretos se pueden citar el proyecto Eye Pet de SONY [14], y el *Bing Maps with augmented reality* de Microsoft [15].

Desafortunadamente existen pocos *frameworks* libres que permitan generar fácilmente aplicaciones de RA. Normalmente se adapta, con considerable esfuerzo, alguno ya existente; tal es el caso de la adaptación de ARToolkit Plus con el motor gráfico Ogre [19].

También debe señalarse que una característica que poseen en común la mayoría de las aplicaciones de RA presentadas por la comunidad científica es la baja complejidad de las mismas. En general consisten en ejemplos básicos diseñados con el propósito

de demostrar el funcionamiento de la librería de RA subyacente, no poniendo explícitamente de manifiesto las posibilidades que puede brindar la RA en situaciones reales.

Esto motivó la creación de un *framework* o ambiente de desarrollo de aplicaciones de RA de código abierto (accesible a todo el mundo), simple e intuitivo, en el que se integran distintas técnicas de RA sobre un motor gráfico extensible revestido con tecnologías gráficas avanzadas. Este *framework* permite la generación de aplicaciones de RA de mediana/alta complejidad en las que se integran fácilmente estas tecnologías.

En este trabajo se detallarán, en primera instancia, cuál es el trabajo relacionado realizado hasta el momento. Luego se describirán los objetivos que se plantearon y que motivaron el desarrollo del *framework* y cómo el mismo está constituido. Posteriormente se presenta un caso de estudio que tiene el objetivo esencial el testeo de cómo funciona el *framework*. Finalmente, la última sección está destinada a presentar los resultados obtenidos.

2 Antecedentes

En el año 2006, en la Universidad de Columbia, se comenzó a desarrollar Goblin XNA [1] un *framework* abierto de RA desarrollado en XNA y C# que soporta varios dispositivos y tecnologías.

La versión de Goblin XNA inicialmente disponible tenía muchas falencias, carecía de una buena legibilidad y era funcionalmente limitado; en el año 2009 publicaron una nueva versión en CodePlex que está significativamente mejor estructurada, es más legible y funcionalmente más completa que las versiones predecesoras. Sin embargo, este *framework* es funcionalmente limitado en ciertos aspectos que consideramos importantes, y su extensibilidad está restringida debido a su estructuración base.

3 Nuestra propuesta

El objetivo general de nuestra propuesta consiste en la creación de un *framework* de desarrollo de aplicaciones gráficas que posibilite integrar fácilmente tecnologías gráficas de alta complejidad, así como la generación de aplicaciones de RA en las que se integren estas tecnologías.

Para alcanzar este objetivo general se determinaron un conjunto de características que debía poseer el *framework*, siendo las más relevantes las que se detallan a continuación:

- **Funcional**: El desarrollador de la aplicación de RA deberá poder contar inmediatamente y sin complicaciones con un amplio conjunto de herramientas y tecnologías. En el diseño del ambiente se priorizó tecnología/inmersión sobre performance, es por esto que los dispositivos móviles no están considerados inicialmente en este proyecto.

- **Abierto**: El *framework*, además de permitir la generación sencilla de aplicaciones, tiene propósitos educativos; por esto es deseable tener acceso al código fuente del mismo.
- **Extensible**/**modificable**: Se deberá permitir la incorporación de nuevas tecnologías y también permitir modificar y extender las ya existentes
- **Eficiente**: El cálculo de RA es por naturaleza demandante, y si se tiene como objetivo la creación de aplicaciones complejas, que incorporan *shaders* y modelos de alta complejidad, la eficiencia del mismo pasa a ser un factor crítico.
- **Fácil de utilizar**/**estructurado**: El *framework* deberá poder ser utilizado por personas que tengan amplios conocimientos de programación gráfica, como por quienes sólo dispongan de conocimientos de programación menos especializados.

3.2 Estructura del *framework*

En lo que respecta al núcleo del *framework*, pueden tenerse en cuenta dos alternativas. La primera de ellas consiste en utilizar un motor gráfico cerrado ya existente de libre uso; la segunda alternativa consiste en utilizar un motor gráfico abierto; éste podría ser uno creado específicamente para este *framework*, o también podría utilizarse un motor gráfico ya existente, como el popular Ogre3D [18]. Las ventajas de esta segunda alternativa son la flexibilidad, la capacidad de disponer de un control más profundo, ideal para la integración de las distintas técnicas de RA y la posibilidad de explorar el código fuente del motor para permitir/potenciar el aprendizaje de cada una de sus partes.

Esta segunda alternativa, desarrollada correctamente, permite que el *framework* reúna todas las características requeridas, particularmente la de ser abierto, siendo esto determinante al momento de optar por esta alternativa. Los motores gráficos abiertos existentes suelen estar programados en C++ y son bastante complejos, lo que dificulta incorporar las distintas tecnologías de RA, y además disminuye su facilidad de uso. Por lo tanto se decidió desarrollar un motor gráfico; éste se implementó en el lenguaje C# utilizando la API gráfica XNA [3]. C# y XNA son populares, potentes, simples y brindan muchas facilidades al desarrollador como la administración de memoria y recursos.

El diseño del *framework* resultante, por lo tanto, consiste en jerarquías de clases que modelan los distintos componentes, es decir, clases relacionadas directamente con la RA, clases para el motor gráfico, clases para los dispositivos de entrada, clases para el sonido y clases con funcionalidades útiles como cronómetros, *pickers*, elementos de interfaz de usuario, etc.

El desarrollador de la aplicación gráfica partirá de una clase destinada a la *lógica* de la aplicación, en la cual definirá qué tareas se realizarán en la carga, actualización, renderizado y terminación de la aplicación. Si bien el desarrollador tiene una gran libertad de acción, el *framework* administrará recursos y se encargará automáticamente de una gran cantidad de tareas, brindando al mismo tiempo un rico conjunto de herramientas y/o posibilidades de acción.

Las clases que se brindan se complementan con la utilización de la herramienta gratuita de creación de shaders FX Composer [4] y la herramienta gratuita de

creación de contenido digital XSI Mod Tool [5]. Cabe aclarar que ninguna de estas herramientas es imprescindible para el desarrollo sobre el *framework*, pero son las herramientas que mejor se adecúan para la creación de *shaders* y modelos respectivamente.

4 Soporte de Realidad Aumentada en el framework

A continuación se detallará qué dispositivos de interacción y librerías de RA fueron necesarios incorporar al *framework* para permitir la generación de aplicaciones de RA; se explicará el por qué de esta elección, y se explicará cómo se realizó la integración de las librerías de RA al *framework*.

4.1 Dispositivos de Interacción. Dispositivos de entrada y de salida.

Un elemento esencial en un entorno de RA es el hardware de interacción, es decir, el conjunto de dispositivos que permitirá la comunicación con el mundo real.

Las técnicas de RA que utilizan cámaras como dispositivo de entrada, en especial los sistemas de marcadores del tipo fiducial, nos presentan un marco de trabajo muy atractivo. Particularmente se seleccionaron estos dispositivos por las siguientes razones:

- El escaso costo de las cámaras web.
- La cantidad y calidad de librerías disponibles para su manipulación, y el soporte comercial y no comercial con el que se cuenta.
- La calidad actual de las librerías de RA que hacen uso de estos dispositivos.

Los sistemas de RA no sólo están basados en la utilización de marcadores de tipo *fiducials*. El *Wii remote* de Nintendo [12], más conocido como *Wiimote*, es un dispositivo que permite, gracias a su cámara infrarroja, triangular su posición o la posición de algún objeto con respecto al mundo. En este sentido podemos destacar el trabajo de Jonny Chung Lee [20] que utilizó el Wiimote para crear aplicaciones innovadoras de RA y Realidad Virtual.

Nuestro *framework*, además de suministrar soporte para teclado, mouse, y xinput gamepads, permite obtener información de hasta 4 Wiimotes, incluyendo la información de sus acelerómetros y su cámara de luz infrarroja. Esto posibilita la implementación de aplicaciones de RA que utilicen Wiimotes como dispositivos de interacción.

El dispositivo de salida (display) seleccionado es el monitor. En esta primera etapa no se experimentará con tecnologías de display innovadoras. Las dos razones que motivan esta decisión son:

- La dificultad de que el usuario disponga de este hardware comercialmente y el costo del mismo.
- La necesidad de invertir el tiempo y el esfuerzo humano en otras áreas que se consideran de mayor relevancia para este proyecto.

Por lo tanto no se tomaron consideraciones específicas referentes a los dispositivos display.

4.2 Librerías de Realidad Aumentada

En lo que respecta a la utilización de librerías de RA, se consideraron dos basadas en sistemas de marcadores del tipo fiducial, ARTag [6, 7] y ARToolkit Plus [9]. La elección se basó principalmente en que ambas son libres, están muy bien formuladas y funcionan bien. También se está evaluando la adaptación de una librería similar a éstas, denominada ALVAR [10] dado que incorpora interesantes características tales como ocultar en tiempo real los marcadores fiducial.

4.2.1 Integración de ARToolkit Plus y ARTag

Luego de realizar un trabajo previo de adaptación ambas librerías pudieron integrarse satisfactoriamente al *framework* (Figura 1). A continuación se detalla cómo se realizó la integración de ambas librerías y cuáles fueron los problemas más relevantes en cada uno de los casos.

La integración de ARToolkit Plus se realizó utilizando un wrapper desarrollado por Casey Chestnut [10]. El primer problema encontrado al realizar esta integración está relacionado con la conversión de las matrices al esquema de matrices de XNA.

Adicionalmente debieron configurarse los parámetros de bytes por pixel en las capturas de la cámara web debido a que ARToolkit Plus espera que la información de estas capturas tenga un formato específico. Tanto ARToolkit como el manejador de cámara web pueden ser configurados para recibir o producir esta información de manera compatible. Con ARTag sucede algo similar y debe adaptarse este parámetro en las capturas.

Por último, ARToolkit Plus requiere que la información de la captura esté almacenada en una estructura de datos nativa de C#, mientras que ARTag en una estructura de datos nativa de C++; esto motivó que debieran hacerse conversiones de estructuras.

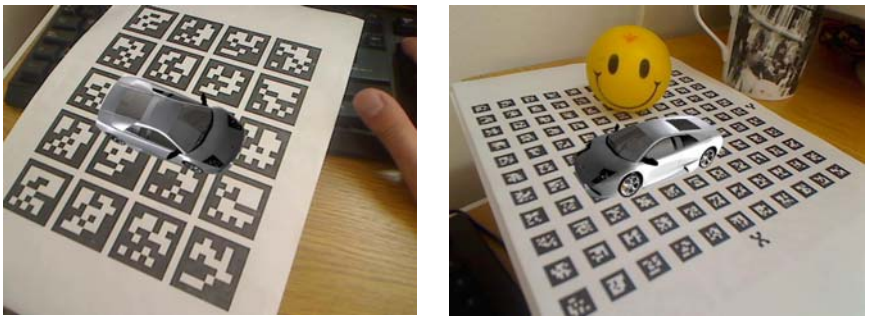

Figura 1. Primeras pruebas de ARToolkit Plus (izquierda) y ARTag (derecha) ejecutándose en el *framework*

ARTag está mejor estructurado que ARToolkit Plus, lo que facilitó parte del proceso de integración. Sin embargo, se presentaron distintos problemas en el transcurso de este proceso; uno de los principales fue ocultar la ventana de OpenGL que utiliza ARTag y reducir el procesamiento de ésta a cero. El wrapper de ARTag [8] utiliza las matrices de OpenGL para devolver los resultados del proceso de

tracking, y desafortunadamente sin una ventana de OpenGL activa no se cargarán dichas matrices con información útil. La solución encontrada fue reducir lo más posible el código de OpenGL, ocultando la creación de la ventana y eliminando todo procesamiento asociada a ella, permitiendo al mismo tiempo utilizar dichas matrices.

5 Gráficos de alta complejidad

En lo que respecta a la facilidad de contar con gráfico de alta complejidad, se trabajó en la incorporación de *shaders* complejos tanto para materiales como para efectos de pre y post procesado. El objetivo es disponer de un marco de trabajo extensible y flexible con shaders, que permitiera al desarrollador utilizar los *shaders* existentes, y también incorporar nuevos. Además, todos los *shaders* incluyen una operación llamada *Test* que permite modificar los parámetros del *shader* en tiempo real a través de la interfaz de usuario provista. Los principales *shaders* actualmente disponibles son *screen space ambient oclussion* (SSAO), en sus variantes *Horizon Based* y *Ray Marching*, *shadow maps*, *blur*, materiales como blinn, constante, Cook-Torrance, *parallax mapping* y un material que simula un océano. En la Figura 2 se puede apreciar gran parte de estos *shaders*.

Asimismo, el *framework* brinda la posibilidad de cargar modelos tridimensionales desde archivos (.x, .xsi y fbx), crear primitivas como cajas y esferas, crear líneas (tanto en espacio de pantalla como en el espacio de mundo), y trabajar con fuentes. También permite generar los *bounding volumes* de los objetos, asociar animaciones simples a éstos, y agrupar un conjunto de objetos en contenedores para operarlos como si se tratase de uno solo. Dispone de un conjunto de cámaras, ofrece *pickers* que utilizan el esquema de textura, capturadores de pantalla, un manejador de *sprites*, y una simple interfaz de usuario. Adicionalmente permite reproducir videos, sonido (posicional y no posicional) y música.

6 Comparación de nuestro *framework* con Goblin XNA

El motor gráfico de Goblin, aunque con características interesantes y ausentes en los ambientes libres de desarrollo de RA hasta su aparición, es parcialmente limitado en varios aspectos que nosotros consideramos importantes. La mayoría de las tecnologías gráficas de las que dispone Goblin fueron tomadas directamente y casi sin variantes del proyecto abierto *Racing Game* de Benjamin Nitschke [2, 13]. A pesar de contar con algunos agregados muy interesantes tales como una simple pero completa *UI* gráfica, manejo de red, etc., deja de lado un esquema robusto de shaders, un manejo cuidadoso de objetos gráficos (por ejemplo carece de un esquema robusto de transformaciones) y de jerarquías, y un tratamiento del núcleo motor que permita tener un alto grado de control y estabilidad del mismo. En lo que respecta a nuestra propuesta, todos estos aspectos fueron especialmente considerados.

Estas carencias se deben principalmente a que *Racing Game*, el proyecto en el que se basa Goblin XNA tiene un propósito muy específico; por esto, su motor gráfico es poco flexible. Goblin XNA replanificó parte de su estructura, pero dejó de lado la reestructuración y la extensión de varios de sus elementos (sistema de materiales, manejo de objetos, el núcleo del motor, etc.) hacia un modelo de organización

presente comúnmente en motores gráficos generales y flexibles, tal como el usado en Ogre y en nuestro *framework*. Este problema afecta principalmente la *funcionalidad* y la *extensibilidad*. En lo referido a la funcionalidad ofrece, por ejemplo, un conjunto muy reducido de materiales y, en cuanto a la extensibilidad no tiene, por ejemplo, una estructura bien formulada que permita agregar nuevos materiales con características significativamente heterogéneas.

Goblin soporta varios dispositivos no convencionales ausentes en nuestra propuesta, como GPS, e iWear, pero carece de soporte para gamepads y Wiimotes. Cabe señalar que nuestra propuesta permitiría incorporar estos dispositivos de manera relativamente sencilla debido a la estructura base del *framework*.

En lo que respecta a librerías de RA la disponibilidad es similar en ambos *frameworks*, Goblin soporta ARTag y ALVAR, mientras que nuestra propuesta soporta ARTag y ARToolkit Plus, y ya se está trabajando en incorporar ALVAR.

7 Caso de estudio

Conjuntamente con el desarrollo del *framework* se trabaja en la generación de una aplicación de prueba (Figura 2) que tiene el objetivo esencial de testear cómo funciona el *framework*, brindándonos la posibilidad de optimizar, mejorar y extender las diferentes partes del mismo. Esta aplicación consiste en una escena con modelos y *shaders* complejos, que pueden ser vistos directamente en el monitor, o pueden ubicarse en el mundo real a través de uno de los sistemas de marcadores fiducial que el usuario seleccione.

Figura 2. Captura de pantalla de la aplicación de prueba en su estado actual, sin la activación de RA. El modelo del auto contiene 500.000 triángulos, y muestra la utilización de Horizon Based SSAO, sombras del tipo *Shadow Mapping* y materiales tipo *Car Paint*, entre otros.

De este modo se pretende aplicar conjuntamente una gran variedad de tecnologías, funciones, *shaders* y modelos con el objetivo de generar una escena digital

convincente y al mismo tiempo sobrecargar la aplicación para lograr optimizarla y liberarla de *bugs* no visibles cuando se trabaja en aplicaciones de poca carga.

7.1 Máquinas de prueba y método de medición

Se evaluó el funcionamiento del *framework* utilizando la aplicación de prueba que se detalló previamente. Para obtener mediciones concretas se ideó un experimento que consiste en evaluar la performance y la estabilidad de la aplicación de prueba en dos computadoras con marcadas diferencias, siendo ambas accesibles para un usuario promedio. Adicionalmente se pretende comparar el desempeño de las librerías de RA, y cómo varía el nivel de interacción variando la resolución de las capturas de la cámara web.

	Computadora 1	**Computadora 2**
GPU	ATI Radeon 4770	GeForce 8800GT
CPU	Intel Celeron Single Core de 3GHz	AMD Phenom II 965 Quad Core corriendo a 3,8 GHz
Memoria RAM	2 Gigas DDR2 de 533Mhz	4 Gigas DDR3 corriendo a 1400Mhz
Sistema Operativo	Windows Vista 32 Bits	Windows 7 64 Bits

La prueba consiste en mostrar una escena *ad hoc* con una configuración gráfica establecida, tratando de identificar con ambas librerías de RA un arreglo de marcadores y posicionando los modelos de la escena en estos marcadores de manera similar en ambas alternativas (Figura 1).

La principal medida que se utilizó para analizar la performance de nuestro proyecto fue la de los cuadros por segundo (FPS). Históricamente, la medición de FPS demostró ser el método más efectivo y sencillo para tomar las mediciones principales de las distintas aplicaciones gráficas. Las razones son sencillas: se puede medir en tiempo real, es de simple comprensión, se conoce cuáles son los límites tolerables para un ojo humano, no entorpece o varía la tarea subyacente, es fácil de implementar y no representa una sobrecarga para la aplicación.

7.2 Resultados obtenidos

En las pruebas realizadas, ARToolkit Plus obtuvo una mejor tasa de FPS que su contraparte ARTag. Sin embargo, a simple vista se puede apreciar que la ganancia en precisión justifica, bajo nuestro criterio, el costo de procesamiento de ARTag. Con ARToolkit Plus en ambas computadoras podríamos ejecutar la aplicación de prueba con la cámara web configurada a una resolución de 800x600 pixeles obteniendo una buena interactividad. Sin embargo, con ARTag en la primera computadora sólo se pueden obtener buenos resultados a una resolución de 320x240 pixeles. En cambio, en la segunda y más poderosa obtenemos buenos resultados a 800x600 pixeles.

Por último, es importante resaltar que el desempeño de estas librerías de RA varía significativamente dependiendo de la carga de la aplicación. Los resultados de las pruebas con escenas de alta complejidad y con configuraciones gráficas demandantes cambian completamente con respecto a los resultados de las pruebas de baja carga, permitiéndonos descubrir problemas de optimización que de otra manera no se podrían apreciar.

7.3 Una experiencia de campo sobre el *framework*

En el primer cuatrimestre del ciclo 2010, dos grupos de alumnos de la materia Computación Gráfica, perteneciente al plan de Ingeniería en Sistemas de Computación de la Universidad Nacional del Sur, realizaron el trabajo final de la materia utilizando este *framework*. Este trabajo consiste en la creación de un juego utilizando los conceptos aprendidos en la materia e incorporando además el uso de dispositivos no convencionales, *shaders* no vistos previamente, sonido, música, etc. Los alumnos contaban con un tiempo de desarrollo acotado a 3 semanas y tenían fecha fija de entrega. Además se contempla que los alumnos están cursando paralelamente otras dos materias.

Ambos grupos estaban formados por tres integrantes y los juegos elegidos fueron un juego de disparos y un juego de autos, utilizando en ambos casos al Wiimote como dispositivo de entrada. El primero utilizó el Nunchuck para los movimientos de traslación del personaje, y la cámara infrarroja para apuntar; el segundo utilizó los acelerómetros para direccionar el volante. Cabe aclarar que se decidió suministrar la mayor parte de los recursos gráficos utilizados en estos proyectos (modelos y texturas) para evaluar mejor el desempeño en programación de los grupos sobre el *framework*.

A pesar de que estos juegos no utilizaron RA, ambos permitieron testear el *framework*, y sobre todo, tener una buena realimentación de su facilidad de uso.

Figura 3. Captura de pantalla del juego de autos (izquierda) y del juego de disparos (derecha)

8 Conclusiones

Los resultados parciales obtenidos son muy satisfactorios. Todas las librerías de RA y las tecnologías gráficas seleccionadas han podido ser integradas manteniendo la base del *framework* simple e intuitiva, lo que muestra su extensibilidad.

Al mismo tiempo, la aplicación de prueba y los dos trabajos realizados en la materia Computación Gráfica nos permiten apreciar la potencialidad del *framework.* Además, estas experiencias refuerzan lo expresado sobre la facilidad de uso, flexibilidad y funcionalidad del *framework*, que permitió la concreción de estos proyectos en los tiempos acordados, y utilizando tecnologías que, dada su complejidad, no habrían podido usarse de otra manera.

Sin embargo, todavía quedan tareas de depuración y optimización, y algunos agregados menores como sistemas de partículas y *shaders* tales como profundidad de campo y HDR, entre otros.

Referencias

1. Oda, O. y Feiner, S., *Goblin XNA*, http://www1.cs.columbia.edu/graphics/projects/goblin/goblinXNA.htm.
2. Oda O., Lister L. J., White S., y Feiner S., *Developing an augmented reality racing*, INTETAIN '08, Cancun, Mexico, 8 - 10 Enero, 2008.
3. Microsoft, *XNA*, http://www.xna.com.
4. NVIDIA, *FX Composer*, http://developer.nvidia.com/object/fx_composer_home.html.
5. Autodesk, *Softimage Mod Tool*, http://autodesk.com.
6. Fiala, M., *ARTag, a fiducial marker system using digital techniques*, *Proc. CVPR 2005*, 590-596, 2005.
7. Fiala, M., *ARTag Rev1: Marker Detection*, http://www.artag.net/rev1.html.
8. Cawood S. y Fiala M., *Augmented Reality. A practical Guide*, ISBN: 978-1-93435-603-6, 2008.
9. Wagner D., Schmalstieg D., *ARToolKitPlus for Pose Tracking on Mobile Devices*, Proc. 12th Computer Vision Winter Workshop, Febrero 2007.
10. VTT Technical Research Centre of Finland, *ALVAR – A Library for Virtual and Augmented Reality*, http://virtual.vtt.fi/virtual/proj2/multimedia/alvar.html
11. Chestnut, C., *Augmented Reality with Windows Presentation Foundation*, http://www.mperfect.net/wpfAugReal.
12. Nintendo, *Wii*, http://wii.nintendo.com.
13. exDream, *XNA Racing Game*, http://www.xnaracinggame.com.
14. SONY, *EyePet*, http://www.eyepet.com.
15. Microsoft Research, *Bing Maps with augmented reality*, http://research.microsoft.com.
16. Martig, S., Castro, S., Fillottrani, P. & Estévez, E **,** *Un Modelo Unificado de Visualización*. Proceedings, pp. 881-892, 9º Congreso Argentino de Ciencias de la Computación. 6 al 10 de Octubre de 2003. La Plata. Argentina.
17. Bimber O. & Raskar R., *Modern Approaches to Augmented Reality*. Conference Tutorial Eurographics, 2004.
18. Ogre, *Ogre3D*, http://www.ogre3d.org.
19. Ogre, *Ogre and ARToolkit Plus*, http://www.ogre3d.org/forums/viewtopic.php?t=22584.
20. Chung Lee J., *Wii Projects*, http://johnnylee.net/projects/wii.